Coerência num sistema transaccional de replicação de objectos em grande escala

João Martins
Ricardo Almeida
Hugo Miranda
Luís Rodrigues

DI–FCUL TR–01–13

Dezembro 2001

Departamento de Informática
Faculdade de Ciências da Universidade de Lisboa
Campo Grande, 1749–016 Lisboa
Portugal

Technical reports are available at `http://www.di.fc.ul.pt/tech-reports`. The files are stored in PDF, with the report number as filename. Alternatively, reports are available by post from the above address.

Coerência num sistema transaccional de replicação de objectos em grande escala

João Martins Ricardo Almeida Hugo Miranda
Luís Rodrigues

Universidade de Lisboa
{jmartins,ralmeida,hmiranda,ler}@di.fc.ul.pt

Dezembro 2001

Resumo

Os sistemas de bases de dados orientados por objectos têm vindo a demonstrar-se uma solução adequada para variadas situações. Neste artigo apresentamos dois protocolos de coerência para bases de dados replicadas em grande escala orientadas a objectos.

1 Introdução

O projecto GlobData tem como objectivo construir um sistema de base de dados distribuído em grande escala orientado por objectos, denominado COPLA. Por forma a aproveitar as bases de dados tradicionais já existentes, o COPLA foi concebido como uma camada de software a funcionar sobre uma base de dados relacional. Os dados são apresentados sob a forma de objectos persistentes replicados, e armazenados em forma relacional.

O uso de primitivas de comunicação em grupo (difusão atómica / virtualmente síncrona) como base para a construção de sistemas de bases de dados replicadas ajuda a lidar com os problemas de tolerância a faltas e gestão de trincos, oferecendo um conjunto de primitivas semanticamente fortes, que permitem uma concretização simples de modelos de coerência fortes [8].

Neste artigo descrevemos dois algoritmos de coerência, desenvolvidos para o sistema COPLA. Estes algoritmos foram concebidos para aproveitar as possibilidades das primitivas de comunicação em grupo, e tendo em mente os objectivos do COPLA, nomeadamente a flexibilidade e escalabilidade do sistema.

Este artigo está organizado da seguinte forma: na secção 2 é referido o trabalho de outros autores nesta área. A secção 3 descreve a arquitectura do COPLA. Na secção 4 são apresentados os algoritmos. A secção 5 conclui este artigo e propõe possíveis direcções de trabalho futuro.

2 Trabalho relacionado

Trabalhos anteriores sobre a construção de bases de dados replicadas, propuseram variados algoritmos para garantir a correcção de transacções replicadas. Alguns traba-

lhos [4, 3] propuseram esquemas de votação, em que é atribuído um certo número de *votos* a cada servidor, e uma transacção apenas pode prosseguir se obtiver um *quorum* de réplicas suficiente. Este quorum tem de ser definido de tal forma que transacções em conflito sejam detectadas em pelo menos uma réplica.

Vários problemas de escalabilidade com este e outros esquemas de replicação foram apontados em [5], onde os autores referem que o número de interbloqueios cresce na proporção n^3 para n nós, e propõem um esquema de dois níveis para ultrapassar estas limitações. Este esquema é uma versão modificada do esquema mestre-escravo: cada objecto pertence a um nó mestre, e para evitar problemas de reconciliação, os nós que não são donos de um objecto fazem actualizações provisórias, contactando de seguida o nó mestre de um objecto para confirmar essa actualizações.

Recentemente a replicação activa ganhou um novo impulso com a introdução de algoritmos de replicação baseados em difusão atómica [10, 6, 9], que utilizam a ordenação oferecida pela difusão atómica para ordenar transacções em conflito.

O projecto Dragon[1] [12, 11, 7] desenvolve esquemas de replicação de bases de dados, aproveitando o trabalho recente da comunidade de sistemas distribuídos em protocolos de difusão atómica de alto desempenho. Usando as propriedades de ordem total e tolerância a faltas, foram concebidos algoritmos de replicação melhorados. Estes algoritmos tiram partido destas propriedades, tornando possível a utilização de esquemas de replicação simples, mas sem as desvantagens apontadas acima, como o número elevado de interbloqueios.

3 O projecto GlobData

O projecto GlobData propõe-se construir uma camada de software que ofereça um serviço de replicação de objectos persistentes transparente para o utilizador. As réplicas estarão distribuídas por uma área geográfica bastante grande (vários países), pelo que os algoritmos a utilizar terão de ter este factor em consideração, devido à elevada latência das comunicações.

3.1 Arquitectura do sistema COPLA

A figura 1 apresenta a arquitectura do sistema COPLA, que é dividida em três grandes módulos ou camadas.

O módulo interface cliente constitui a face visível para o programador do sistema GlobData. É responsável por oferecer uma interface que permita desenvolver aplicações finais, e comunicar com os outros dois módulos (protocolos de coerência e UDS (Uniform Data Store)) para fornecer os seus serviços.

O módulo "protocolos de coerência" é responsável por garantir que os dados nas diferentes réplicas se mantêm sincronizados. Os diferentes protocolos de coerência disponibilizam uma interface uniformizada (denominada CP-API na figura) ao nível acima. Isto permite que o COPLA seja configurado de acordo com as características do sistema.

O módulo "armazenamento de dados uniforme" é responsável por traduzir o modelo apresentado ao programador de objectos persistentes em conjuntos de tabelas a serem armazenados num SGBD relacional tradicional. É também responsável por converter todas as operações solicitadas pela aplicação utilizando um subconjunto de Object Query Language [2] (OQL) em interrogações SQL normalizadas.

[1] `http://www.inf.ethz.ch/department/IS/iks/research/dragon.html`

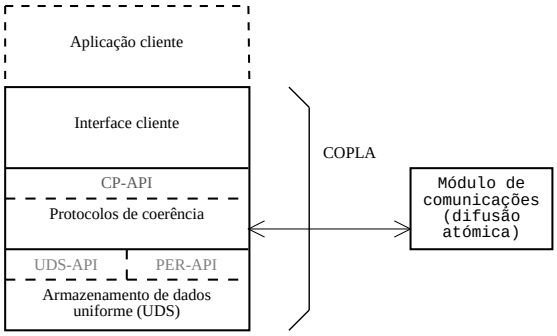

Figura 1: Módulos COPLA

3.2 Desafios da arquitectura

Os algoritmos apresentados estendem os desenvolvidos em trabalhos anteriores, de forma a satisfazer os objectivos do projecto GlobData, que se distingue pela conjugação dos seguintes requisitos:

- Grande escala: os trabalhos anteriores em sistemas de bases de dados replicadas utilizavam servidores interligados por uma rede local dedicada para o efeito. O projecto GlobData pretende construir um sistema geograficamente disperso em que os nós comuniquem através da Internet.
- Independência do gestor de bases de dados: os trabalhos existentes integram os protocolos com o motor de bases de dados. Tal aproximação tem vantagens, já que permite um controlo total sobre o desenrolar das transacções, mas obriga a modificar o núcleo do gestor da base de dados.
- Possibilidade de mudança dos protocolos: o sistema COPLA deve ser adaptável a diversas realidades de escala do sistema e facilidades de comunicação. Deve por isso suportar a possibilidade de utilizar diversos protocolos de coerência, que poderão apresentar desempenhos distintos.
- Orientação a objectos: dado que o sistema COPLA irá projectar objectos em relações, esta operação deverá ser isolada dos protocolos de coerência, por forma a não ligar os algoritmos de coerência com uma determinada forma de representação dos objectos.

4 Dois algoritmos de coerência para o COPLA

Estes algoritmos de coerência foram desenhados tendo em conta os objectivos do projecto GlobData. O facto de o sistema funcionar em grande escala levou a que fosse feito um esforço para limitar a largura de banda utilizada e levar em conta a latência elevada, através da optimização que denominámos “actualizações diferidas”, descrita mais à frente.

A definição de uma interface comum aos protocolos e a existência do módulo de armazenamento de dados independente permitiu que os protocolos fossem intercambiáveis.

A cada objecto constante da base de dados é atribuído um identificador único, designado de OID (*object identifier*). Os protocolos de coerência lidam apenas com estes identificadores, considerando opacos os dados do objecto em si. Isto permite que a representação dos objectos na base de dados possa variar sem que isso se repercuta no protocolo.

Modelo transaccional do COPLA No COPLA, uma transacção desenrola-se da seguinte forma:

1. O programador assinala que quer iniciar uma transacção.
2. O programador faz uma interrogação à base de dados, usando um subconjunto da linguagem OQL. Esta interrogação devolve uma colecção de objectos.
3. Os objectos devolvidos são manipulados pelo programador através de funções do COPLA. Estas funções permitem obter ou modificar os valores dos atributos de cada objecto, e obter novos objectos através das relações (atributos que são referências para outros objectos).
4. Os passos 2-3 são repetidos até que a transacção esteja completa.
5. O programador pede ao sistema para confirmar a transacção.

Interacção com os protocolos de coerência As operações executadas pela aplicação dão origem a três pontos de interacção com o protocolo de coerência. Para cada um desses três pontos foi definida uma chamada na interface:

- Interrogações à base de dados – evaluate().
- Acessos directos Aplicação – UDS – getNewVersions().
- Finalização (confirmação/cancelamento) de uma transacção – commit()/abort().

Os dois primeiros pontos cobrem todas as formas de leitura de objectos: as interrogações e o acesso por seguimento de relações entre objectos. As alterações aos objectos são mantidas pela UDS. Em tempo de finalização de uma transacção, o protocolo pede à UDS as alterações efectuadas, que as fornece sob a forma de uma lista de OIDs dos objectos alterados e um conjunto de dados opaco, contendo as alterações em si. Desta forma o conjunto de leituras e escritas de uma transacção é totalmente conhecido no momento de confirmação.

As estratégias de replicação de uma base de dados distribuída podem ser classificadas de acordo com três parâmetros [11]. Os dois protocolos apresentados podem ser classificados como actualização global com interacção constante. Actualização global, porque todos os nós actualizam a sua cópia dos dados quando uma transacção é executada. Esta aproximação foi escolhida devido à sua resistência a faltas (já que todos os servidores mantêm cópias actualizadas dos dados) e à não criação de pontos de engarrafamento, como a técnica de cópia primária. A interacção é constante, porque o número de mensagens trocadas por transacção é constante, independentemente do número de operações que cada transacção contém. Esta opção é nitidamente melhor do que interacção linear, já que os custos de comunicação são elevados. Os protocolos descritos abaixo exploram o terceiro grau de liberdade: o modo de terminação das transacções (com ou sem votação).

Interacção com o protocolo de difusão atómica Um protocolo de difusão atómica assegura a difusão de mensagens por um grupo de servidores, garantido atomicidade e ordem de entrega das mensagens. Mais detalhadamente, sejam m e m' duas mensagens enviadas por difusão atómica para um grupo de servidores g. A entrega é *atómica* se quando um membro de g entregar m (resp. m'), então todos os membros de g entregam m (resp. m'). A entrega é *ordenada* se quando dois membros de g entregarem m e m', o fazem pela mesma ordem.

As duas propriedades destes protocolos de comunicação podem ser usadas como ponto de partida para a construção de um algoritmo de seriação de transacções distribuído: a garantia de ordem é um mecanismo poderoso para a resolução de conflitos, e a garantia de entrega torna os algoritmos mais simples, já que não são necessárias fases de confirmação adicionais.

4.1 O protocolo sem votação

O protocolo sem votação é uma modificação do algoritmo descrito em [10], alterado para efectuar o controlo de concorrência utilizando versões [1] em vez de trincos, e adaptado ao modelo transaccional usado no COPLA.

Este algoritmo mantém, para cada objecto, o seu número de versão, e se o objecto está ou não actualizado (isto é, se a versão constante da base de dados é ou não a mais recente). Caso o objecto esteja desactualizado, é mantido também o identificador de um nó que detém a versão mais recente. Esta informação, denominada tabela de consistência, é mantida em armazenamento persistente, e é alterada no contexto da mesma transacção que altera os dados (i.e., a informação de consistência só é alterada se a transacção for confirmada).

Quando um objecto é criado, o seu número de versão é posto a zero. De cada vez que uma transacção actualiza um objecto, e essa transacção é confirmada, a versão do objecto é incrementada. Este mecanismo mantém os números de versão dos objectos sincronizados, uma vez que a ordem total da difusão das mensagens é assegurada, logo todos os nós processam as transacções pela mesma ordem.

Duas transacções t e t' estão em conflito se t' leu um objecto o com número de versão v_o, e em tempo de confirmação de t', o número de versão de o constante na base de dados, v'_o é superior a v_o. Isso significa que t' leu dados que foram subsequentemente modificados (por t, que modificou o objecto, terminou antes de t' e incrementou o número de versão), devendo por isso t' ser abortada.

Descreve-se de seguida de uma forma geral o funcionamento do protocolo de resolução de conflictos sem votação.

1. Todas as acções de uma transacção são executadas localmente no nó onde foi iniciada a transacção (nó delegado).

2. Quando a aplicação tenta confirmar a transacção, o conjunto de leituras e o conjunto de escritas são disseminados usando difusão atómica.

3. Quando uma transacção é recebida da difusão atómica, todos os servidores verificam se esta não entra em conflito com outras transacções. Não entrar em conflito significa que as versões dos objectos lidos pela transacção recém-chegada são maiores ou iguais às versões desses mesmos objectos constantes na base de dados local.

- *evaluate(t):*
  1. *Obter da UDS a lista l de objectos que a interrogação necessita.*
  2. *Invocar getNewVersions(t,l), assegurando que a cópia local desses objectos está actualizada.*
  3. *Passar a interrogação à UDS, que a executará na base de dados local e devolverá a lista de objectos resultante.*
- *commit(t):*
  1. *Obter a lista de objectos lidos (RS_t) e escritos (WS_t) da UDS.*
  2. *Comunicar $< t, RS_t, WS_t >$ por difusão atómica.*
  3. *Quando a transacção for recebida da difusão atómica:*
     (a) *Verificar se a transacção entra em conflito. Se não, abortar a transacção. Se sim, prosseguir.*
     (b) *Abortar todas as transacções em conflito com a transacção recebida.*
     (c) *Confirmar a transacção.*

Figura 2: Protocolo sem votação

Se a transacção não está em conflito, então é confirmada, se está é abortada. Dado que este procedimento é determinista e todos os nós, incluindo o nó delegado, recebem as transacções pela mesma ordem, todos os nós chegam à mesma decisão. O nó delegado pode agora informar o cliente do resultado da transacção.

Note-se que o último passo é executado por *todos* os servidores, incluindo aquele que iniciou a transacção.

Apresentamos na figura 2 uma descrição mais detalhada do funcionamento do algoritmo. Este encontra-se dividido em três funções, que correspondem aos três pontos de interacção descritos acima: interrogações à base de dados, acesso a objectos e terminação de transacções. Todas estas funções aceitam um parâmetro, t, a transacção a tratar. A função getNewVersions(t,l) simplesmente acrescenta os objectos da lista l à lista de objectos lidos por t.

Na função commit(), o passo 3 é executado por todos os servidores.

Como se pode verificar, o algoritmo utiliza a ordem total dada pela difusão atómica para resolver conflitos: se uma transacção é entregue pela ordem total e está consistente, tem prioridade sobre as outras. Isto implica que se duas transacções t_1 e t_2 estão em conflito, e t_1 é entregue antes de t_2, então t_1 irá prosseguir, e t_2 será dada como incoerente, uma vez que leu dados antigos que foram alterados.

A decisão é tomada por cada nó, mas todos os nós chegarão à mesma decisão já que esta depende apenas da ordem de entrega das duas transacções, que as propriedades da difusão atómica asseguram ser a mesma em todos os nós.

Note-se que se a aplicação decidir cancelar uma transacção não é necessário efectuar qualquer comunicação. A função commit() não é executada, sendo suficiente libertar os recursos locais utilizados.

4.2 O protocolo com votação

Nesta secção apresentamos o protocolo com votação. Este protocolo resulta de uma adaptação ao modelo transaccional do COPLA do protocolo descrito em [7]. Consiste em duas fases, uma de difusão de escritas, e uma de votação.

O funcionamento geral do algoritmo é o seguinte:

1. A transacção é executada localmente no nó delegado, obtendo trincos de leitura (locais) sobre os objectos lidos.
2. Quando a aplicação pede a confirmação da transacção, o conjunto de escritas da transacção é disseminado por difusão atómica.
3. Quando o conjunto de escritas é recebido, todos os servidores tentam obter os trincos de escrita sobre os objectos constantes no conjunto. Quando o nó que iniciou a transacção obtém todos os trincos, dissemina uma mensagem de confirmação da transacção, por difusão atómica.
4. Quando um servidor recebe uma mensagem de confirmação, aplica as alterações à sua base de dados e liberta todos os trincos correspondentes.

Apresentamos na figura 3 uma descrição mais detalhada do algoritmo. A função getNewVersions(t,l) funciona do modo descrito na secção anterior.

Como se pode observar, o algoritmo usa a ordem das mensagens dada pela difusão atómica para ordenar as transacções em conflito. A ordem final das transacções é dada pela ordem das mensagens $< t, \mathrm{WS}_t >$ e c_t (ver a função commit() da figura 3).

A detecção de conflitos entre transacções é feita utilizando os trincos.

Os conflitos de escrita-escrita, i.e., que ocorrem quando duas transacções concorrentes tentam escrever no mesmo objecto, são detectados pelo sistema de trincos (duas transacções tentam obter um trinco de escrita no mesmo objecto). Dado que os trincos de escrita são obtidos na recepção de $< t, \mathrm{WS}_t >$, a ordem destas mensagens determina a ordem de aquisição dos trincos. Como se pode observar na figura 3, se uma transacção t obtiver um trinco de escrita, forçará uma transacção posterior t' a esperar para obter um trinco de escrita sobre o mesmo objecto. Se t terminar, forçará t' a abortar.

Os conflitos de leitura-escrita, i.e., que ocorrem quando duas transacções concorrentes tentam aceder ao mesmo objecto, uma para o ler e outra para o escrever, são resolvidos dando prioridade às escritas: quando uma mensagem $< t, \mathrm{WS}_t >$ é recebida, os trincos de escrita são adquiridos, provocando o cancelamento de transacções que detenham trincos de leitura nos objectos listados em WS_t.

Este processo de resolução é executado por cada nó, e dado que as mensagens $< t, \mathrm{WS}_t >$ são entregues pela mesma ordem em todos os nós e o processo é totalmente determinista, todos os nós decidem a mesma ordem para as transacções.

4.2.1 Optimização

Este protocolo pode ser melhorado por forma a evitar o cancelamento de transacções em excesso. Com efeito, na fase de obtenção dos trincos de escrita (após a recepção de $< t, \mathrm{WS}_t >$), em vez de abortar imediatamente transacções que detenham trincos de leitura nos objectos constantes em WS_t, estas podem ser colocadas num estado alternativo, denominado "EXECUTING_ABORT", prevenindo situações em que t venha a ser abortada, o que levaria ao cancelamento desnecessário das referidas transacções que detenham trincos de leitura. Assim sendo, transacções no estado "EXECUTING_ABORT" podem prosseguir, mas não podem ser confirmadas. Se o fizerem

- *evaluate(t):*
  1. *Obter da UDS a lista l de objectos que a interrogação necessita.*
  2. *Obter um trinco de leitura para cada objecto em l. Se algum desses objectos está trancado para escrita, a transacção é posta em espera até que esse objecto seja libertado.*
  3. *Invocar getNewVersions(t, l), assegurando que a cópia local desses objectos está actualizada.*
  4. *Passar a interrogação à UDS, que a executará na base de dados local e devolverá os objectos resultantes.*
- *commit(t):*
  1. *Obter a lista de objectos escritos (WS_t) por t da UDS.*
  2. *Comunicar $< t, \mathrm{WS}_t >$ por difusão atómica.*
  3. *Quando for recebida a transacção da difusão atómica:*
     (a) *Tentar obter um trinco de escrita para todos os objectos o constantes em WS_t:*
        i. *Se existir um trinco de leitura em o, a transacção que detém o trinco de leitura é abortada, e o trinco de escrita é concedido a t.*
        ii. *Se existir um trinco de escrita em o, ou todos os trincos de leitura em o são de transacções u cuja mensagem $< u, \mathrm{WS}_u >$ já tenha sido recebida, t fica em espera neste objecto até que esses trincos sejam libertos.*
        iii. *Se não existir mais nenhum trinco em o, conceder o trinco a t.*
     (b) *Se este nó é o nó delegado de t, comunicar c_t por difusão atómica.*
  4. *Quando é entregue uma mensagem c_t: Confirmar a transacção, escrevendo todas as alterações e libertando todos os trincos adquiridos por t. Todas as transacções que estavam à espera de obter um trinco de escrita em algum objecto alterado por t são abortadas.*
  5. *Quando é entregue uma mensagem a_t: Se a transacção é local a mensagem é ignorada, senão aborta t, libertando todos os seus trincos.*

Figura 3: Protocolo com votação

serão postas em espera. Se a transacção t for confirmada, então as transacções que foram colocadas em "EXECUTING_ABORT" são abortadas. Se a transacção t for abortada, então as transacções em "EXECUTING_ABORT" passam de novo ao estado de execução, prosseguindo normalmente.

As transacções só de leitura que tentam terminar e estão no estado "EXECUTING_ABORT" não necessitam de ser colocadas em espera - podem ser terminadas imediatamente. O efeito final é que estas transacções são ordenadas antes da transacção que as colocou em "EXECUTING_ABORT".

4.3 Comparação

Os algoritmos apresentados, embora ofereçam as mesmas garantias, seguem estratégias diferentes:

O algoritmo com votação não difunde o conjunto de leituras da transacção, mas no entanto necessita de mais um passo para confirmar uma transacção.

1. *Para cada OID da lista l:*
   (a) *Verificar se a cópia do objecto correspondente na base de dados está actualizada.*
   (b) *Se o objecto estiver desactualizado, obter a cópia dos dados mais recente do nó que a detém.*
   (c) *Passar o campo "owner" do objecto a vazio.*
2. *Adicionar a lista l à lista de leituras da transacção t.*

Figura 4: getNewVersions(t, l)

O algoritmo sem votação usa apenas uma mensagem de difusão atómica por transacção, mas esta tem de conter o conjunto de leituras e escritas efectuadas, que se pode revelar de grande dimensão.

Existe portanto uma troca entre quantidade e tamanho de mensagens que influencia o desempenho dos algoritmos. Nenhum deles é claramente melhor que o outro, dado que não só a natureza das transacções (com mais ou menos objectos) como as próprias condições variáveis das comunicações determinarão qual a escolha mais acertada. Certamente que os testes a realizar nos protótipos ajudarão a determinar quais as condições óptimas para cada um dos algoritmos.

4.4 Actualizações diferidas

Ambos os algoritmos apresentados suportam uma optimização, denominada "actualizações diferidas", que consiste em atrasar a transferência de dados até que estes sejam solicitados por uma transacção. Isto requer algumas modificações ao algoritmo, na função getNewVersions(t,l), que descrevemos na figura 4.

É necessário existir, associado a cada OID, um campo, denominado "owner", que contém o identificador do nó que detém a cópia mais recente dos dados. Se esse campo se encontrar vazio, significa que é o próprio nó que detém a cópia mais recente dos dados.

Antes da optimização, em ambos os algoritmos a escrita dos dados modificados é efectuada no fim de uma transacção, depois da confirmação. Esse processo é modificado: apenas o nó que iniciou a transacção escreve os dados para disco, colocando o campo "owner" a vazio. Os restantes nós limitam-se a preencher o campo "owner" com a identificação do nó que iniciou a transacção.

5 Conclusões e trabalho futuro

Neste artigo apresentámos dois algoritmos de coerência, desenvolvidos para o sistema COPLA. Estes algoritmos foram concebidos para aproveitar as possibilidades das primitivas de comunicação em grupo, e tendo em mente os objectivos do COPLA, nomeadamente a flexibilidade e escalabilidade do sistema.

Este trabalho foi desenvolvido no âmbito do projecto GlobData, que pretende desenvolver um sistema de base de dados replicada em larga escala e orientado por objectos. Dado que o sistema deverá ser escalável, foi desenvolvida uma arquitectura modular, que possibilita a utilização de diversos protocolos de coerência. Neste ar-

tigo apresentámos dois desses protocolos de coerência, com comportamentos distintos conforme o perfil de utilização do sistema.

Ambos os protocolos estão na fase inicial de concretização, pelo que não existe ainda meio de os comparar na prática. No entanto, analisando a sua estrutura é possível conjecturar qual o seu comportamento em casos típicos. Em cargas de trabalho com poucas transacções em conflito, é de prever que o algoritmo sem votação ganhe vantagem, já que utiliza apenas uma mensagem de difusão atómica por transacção. Na situação oposta, com conflitos elevados, o protocolo com votação poderá ser vantajoso, porque abortará um número menor de transacções. Esperamos que os dados que iremos obter dos protótipos ajudem a identificar mais "pontos quentes" nos algoritmos passíveis de optimização. Estes dados ajudarão também a medir o impacto da optimização que denominámos "actualizações diferidas": esta foi pensada tendo em conta que a maior parte das transacções acede a subconjuntos distintos de dados. É importante verificar o desempenho do algoritmo se esta condição não for verdadeira.

Referências

[1] P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.

[2] R. G. G. Cattell. *The Object Data Standard: ODMG3.0*. Morgan Kauffmann Publishers, 2000.

[3] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, October 1985.

[4] D. Gifford. Weighted voting for replicated data. In *actas do 7th ACM Symposium on Operating System Principles*, pages 150–162, Pacific Grove, Califórnia, EUA, December 1979.

[5] J. Gray, P. Helland, P. O'Neal, and D. Shasha. The dangers of replication and a solution. In *actas da 1996 ACM SIGMOD International Conference on Management of Data*, pages 173–182, Montreal, Quebec, Canadá, June 1996.

[6] J. Holliday, D. Agrawal, and A. El Abbadi. Using multicast communication to reduce deadlock in replicated databases. In *actas do IEEE Symposium on Reliable Distributed Systems (SRDS2000)*, October 2000.

[7] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *actas da 18th International Conference on Distributed Computing Systems (ICDCS)*, Amsterdão, Holanda, May 1998.

[8] B. Kemme and G. Alonso. Transactions, messages and events: Merging group communication and database systems. In *actas da 3rd ERSADS European Research Seminar on Advances in Distributed Systems*, Ilha da Madeira, Portugal, April 1999.

[9] M. Patiño-Martínez, R. Jimenez-Peris, B. Kemme, and G. Alonso. Scalable replication in database clusters. In *actas do 14th International Symposium on Distributed Computing (DISC)*, Toledo, Espanha, October 2000.

[10] F. Pedone, R. Guerraoui, and A. Schiper. Exploting atomic broadcast in replicated databases. In *actas da EuroPar (EuroPar'98)*, September 1998.

[11] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Database replication techniques: a three parameter classification. In *actas da 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000)*, Nürnberg, Alemanha, October 2000.

[12] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *actas da 20th International Conference on Distributed Computing Systems (ICDCS)*, Taipei, Taiwan, República da China, April 2000.